Fuzzy Quantified Queries to Fuzzy RDF Databases

Olivier Pivert, Olfa Slama, Virginie Thion
Rennes 1 University / IRISA
Lannion, France
Email: {Olivier.Pivert,Olfa.Slama,Virginie.Thion}@irisa.fr

***Abstract*—In a relational database context, fuzzy quantified queries have been long recognized for their ability to express different types of imprecise and flexible information needs. In this paper, we introduce the notion of fuzzy quantified statements in a (fuzzy) RDF database context. We show how these statements can be defined and implemented in FURQL, which is a fuzzy extension of the SPARQL query language that we previously proposed. Then, we present some experimental results that show the feasibility of this approach.**

I. Introduction

The Resource Description Framework (RDF) [17] is the standard data model promoted by the W3C[1] for representing information about resources available on the Web. SPARQL [16], is the official W3C recommendation for querying these data in a crisp way.

In a previous work [13], we proposed a first extension of SPARQL (called FURQL) aimed to support flexible querying of crisp and fuzzy RDF databases. In a relational database context, fuzzy quantified statements have been long recognized for their ability to express different types of imprecise information needs [1]. Recently, such statements started to attract attention of many researchers [20], [4], [5], [14] in a graph database context. However, in the specific RDF/SPARQL setting, the current approaches from the literature that deal with quantified queries consider crisp quantifiers only [3], [6].

In the present paper, we intend to integrate fuzzy quantified statements in a FURQL query addressed to a fuzzy RDF database. We use as a starting point our previous work [14] where we extended the CYPHER language, used for querying crisp graph databases, with fuzzy quantified structural queries.

The remainder of this paper is organized as follows. Section II presents background notions. In Section III, which is the core of the contribution, we introduce the syntactic format for expressing fuzzy quantified statements in the FURQL language defined in [13] and we describe their interpretation. Section IV deals with query processing and discusses implementation issues. In Section V, some experimental results showing the feasibility of the approach are presented. Related work is discussed in Section VI. Finally, Section VII recalls the main contributions and outlines research perspectives.

II. Background notions

In this section, we recall important notions about both the classical and the fuzzy RDF data models, as well as reminders about the query languages SPARQL and FURQL. We also provide a refresher about fuzzy quantified statements.

[1]World Wide Web Consortium

A. RDF and Fuzzy RDF (F-RDF) Data Models

The Resource Description Framework (RDF) [17] uses pairwise disjoint infinite sets of resource names, literals and blank nodes (i.e., unknown or anonymous resources) respectively denoted by $\mathcal{U}$, $\mathcal{L}$ and $\mathcal{B}$ in the following.

Let us consider a music album as a resource of the Web. A characteristic may be attached to the album, like a title, an artist, a date or a set of tracks. In order to express such a characteristic, the RDF data model uses a statement in the form of a triple $\langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$. The subject **s** denotes the resource being described, the predicate **p** denotes the property of the resource and the object **o** denotes the property value. A triple states that the subject **s** has a property **p** with a value **o**. For instance, the triple ⟨`Beyonce, creator, B'Day`⟩ states that `Beyonce` has `B'Day` as a `creator` property, which can be interpreted as `Beyonce` is a creator of `B'Day`. A set of RDF triples can be modeled by a directed labeled graph (called *RDF graph* or simply *graph* in the following) where for each triple $\langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle$, the subject **s** and the object **o** are nodes, and the predicate **p** corresponds to an edge from the subject node to the object one. RDF is then a graph-structural data model that makes it possible to exploit the basic notions of graph theory (such as node, edge, path, neighborhood, connectivity, distance, in-degree, out-degree, etc.).

Unfortunately, the classical RDF model is only capable of representing Boolean notions whereas real-world concepts are often of a gradual nature. This is why several authors proposed fuzzy extensions of the RDF model. Throughout this paper, we consider the data model based on Definition 1 which synthesizes the existing fuzzy RDF models of literature (like e.g. [11] or [10]), whose common principle consists in adding a fuzzy degree to edges, modeled either by a value embedded in each triple or by a function associating a satisfaction degree with each triple, expressing the extent to which the fuzzy concept attached to the edge is satisfied. For instance, the fuzzy triple (⟨`Beyonce, recommends, Euphoria`⟩, 0.8) states that ⟨`Beyonce, recommends, Euphoria`⟩ is satisfied to the degree 0.8, which could be interpreted as `Beyonce` *strongly* recommends `Euphoria`.

Definition 1 (Fuzzy RDF (F-RDF) graph): An F-RDF graph is a tuple $(\mathcal{T}, \zeta)$ such that (i) $\mathcal{T}$ is a finite set of triples of $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$, (ii) ζ is a membership function on triples $\zeta : \mathcal{T} \rightarrow [0, 1]$.

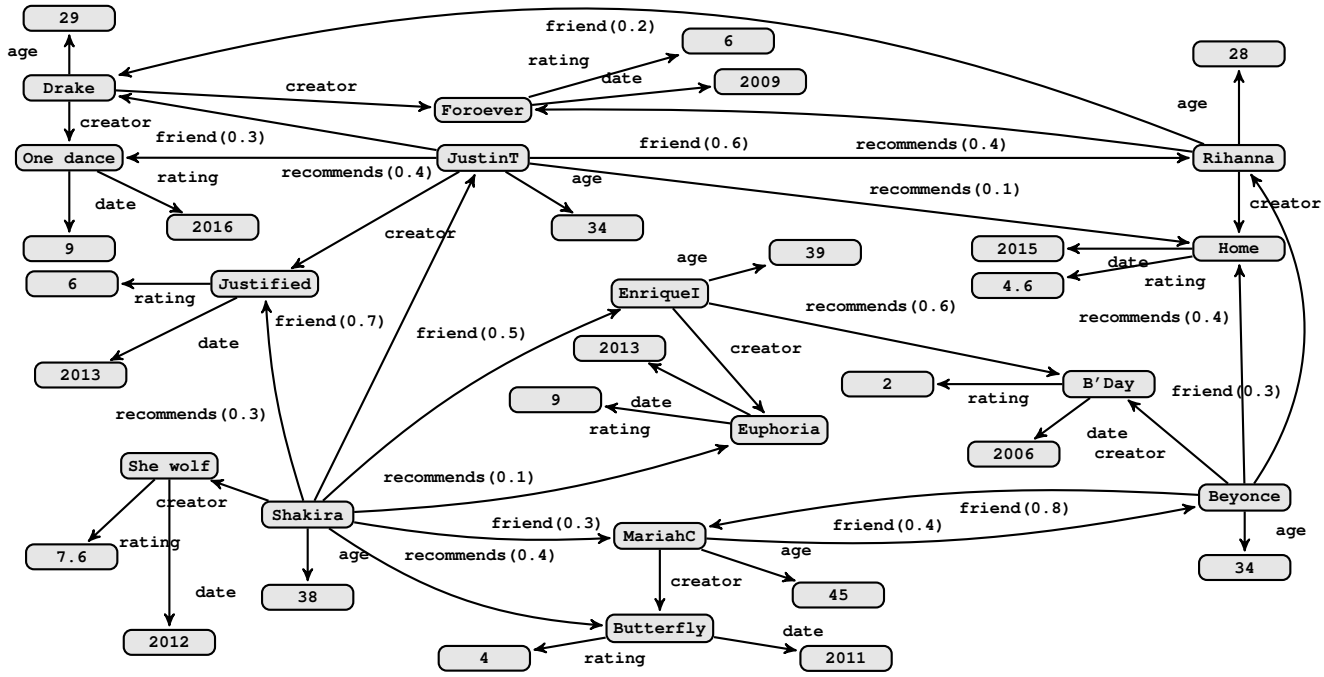


Figure 1. Fuzzy RDF graph G_{MB} inspired by MusicBrainz

According to the classical semantics associated with fuzzy graphs, $\zeta(t)$ qualifies the intensity of the relationship involved in the statement t. Intuitively, ζ attaches fuzzy degrees to edges of the graph. Having a value of 0 for ζ is equivalent to not belonging to the graph. Having a value of 1 for ζ is equivalent to fully satisfying the associated concept. In the graph G_{MB} of Figure 1, such edges appear as classical ones, i.e. with no degree attached. The fuzzy degrees associated with edges can be given or calculated. In its simplest form, each degree may be based on a simple statistical notion, e.g. the intensity of friendship between two artists may be computed as the number of their common friends over the total number of friends that they have.

Example 1: Figure 1 is an example of a Fuzzy RDF graph, denoted by G_{MB} in the following, inspired from MusicBrainz[2]. It mainly contains artists and albums as nodes. For readability reasons, each URI node contains the value of its *name* instead of the URI itself. Literal values may be attached to URIs, like the age of an artist, the release date or the global rating of an album. The graph contains fuzzy relationships (e.g. *recommends*) as well as crisp ones (e.g. *creator*,...).$\diamond$

B. SPARQL and FURQL

SPARQL [16] is the standard query language promoted by the W3C for querying RDF Data. It is a declarative query language based on graph pattern matching, in the sense that the query processor searches for sets of triples in the data graph that satisfy a pattern expressed in the query.

[2]https://musicbrainz.org/

Roughly speaking, a graph pattern is defined as triples where variables can occur, composed by binary operators UNION, FILTER, OPTIONAL and . (concatenation).

Example 2: Listing 1 gives an example of a SPARQL query that retrieves every artist such that the albums he/she recommends have a rating under 4 and were created by another artist connected by a path of *friend* links to him/her. $\diamond$

```
SELECT ?art1 WHERE {
  ?art1 recommends ?alb. ?alb rating ?rating.
  ?art1 friend+ ?art2. ?art2 creator ?alb.
FILTER (?rating < 4) }
```

Listing 1. A SPARQL query

The language FURQL (Fuzzy RDF Query Language) defined in [13], uses fuzzy extensions of the SPARQL graph patterns introduced in [12] in order to express fuzzy preferences on the entities of an F-RDF graph (through fuzzy conditions) and on the structure of the graph (through fuzzy regular expressions). A *fuzzy graph pattern* considers the following binary operators: AND (SPARQL concatenation), UNION (SPARQL UNION), OPT (SPARQL OPTIONAL) and FILTER (SPARQL FILTER).

Syntactically, FURQL extends SPARQL by allowing the occurrence of fuzzy graph patterns in the WHERE clause and the occurrence of fuzzy conditions in the FILTER clause. A fuzzy regular expression is close to a property path, as defined in SPARQL 1.1 [8], and may involve fuzzy structural properties (e.g. concerning the *distance* between two nodes or the *strength* of a path).

Example 3: The FURQL query of Listing 2 retrieves the artists that recommend low-rated albums made by friends or

related friends of friends, and performs an alpha-cut on the answers (only those having a satisfaction degree greater or equal to 0.3 are kept) The CUT clause is of course optional.◇

```
DEFINEDESC low AS (3,6)
SELECT ?art1 WHERE {
  ?art1 recommends ?alb. ?alb rating ?rating.
  ?art1 friend+ ?art2. ?art2 creator ?alb.
FILTER (?rating IS low)
} CUT 0.3
```

Listing 2. A FURQL query

C. Fuzzy Quantified Statements

We first recall important notions about fuzzy quantifiers, then we present different approaches from the literature for interpreting fuzzy quantified statements.

1) Fuzzy Quantifiers: Fuzzy quantifiers makes it possible to model quantifiers from the natural language (*most of*, *at least half*, *few*, *around a dozen*, etc). Zadeh [21] distinguishes between absolute (e.g, *at least three*) and relative (e.g, *most*) quantifiers. Absolute quantifiers refer to a number while relative ones refer to a proportion. An absolute quantifier is represented by a function μ_Q from an integer (or real) range to $[0,\ 1]$ whereas a relative quantifier is a mapping μ_Q from $[0,\ 1]$ to $[0,\ 1]$. In both cases, the value $\mu_Q(j)$ is defined as the truth value of the statement "$Q\ X$ are A" when exactly j elements from X fully satisfy A (whereas it is assumed that A is fully unsatisfied by the other elements).

Calculating the truth degree of the statement "$Q\ X$ are A" raises the issue of determining the cardinality of the set of elements from X which satisfy A. If A is a Boolean predicate, this cardinality is a precise integer (k), and then, the truth value of "$Q\ X$ are A" is $\mu_Q(k)$. If A is a fuzzy predicate, this cardinality cannot be established precisely and then, computing the quantification corresponds to establishing the value of function μ_Q for an imprecise argument.

2) Interpretation of Fuzzy Quantified Statements: We now present different proposals from the literature for interpreting quantified statements of the type "$Q\ B\ X$ are A" (which generalizes the case "$Q\ X$ are A" by considering that the set to which the quantifier applies is itself fuzzy) where X is a (crisp) referential and A and B are fuzzy predicates.

a) Zadeh's Interpretation: Let X be the usual (crisp) set $\{x_1,\ x_2,\ \ldots,\ x_n\}$ and n the cardinality of X. Zadeh [21] defines the cardinality of the set of elements of X which satisfy A, as: $\Sigma count(A) = \sum_{i=1}^{n} \mu_A(x_i)$.

The truth degree of the statement "$\mathcal{Q}\ B\ X$ are A" (with $\mathcal{Q}$ relative) is then given by:

$$\begin{aligned} \mu(\mathcal{Q}\ B\ X\ are\ A) &= \mu_{\mathcal{Q}}\left(\frac{\Sigma count(A \cap B)}{\Sigma count(B)}\right) \\ &= \mu_{\mathcal{Q}}\left(\frac{\sum_{x \in X} \top(\mu_A(x),\ \mu_B(x))}{\sum_{x \in X} \mu_B(x)}\right) \end{aligned} \quad (1)$$

where $\top$ denotes a triangular norm (e.g., the minimum).

b) Yager's Competitive Type Aggregation: The interpretation by decomposition described in [18] is limited to *increasing* quantifiers. The statement "$Q\ B\ X$ are A" is true if there exists a crisp subset C of X that satisfies the conditions (c'_1) $Q\ B\ X$ are in C and ($c'2$) each element x of C satisfies the implication $(x \text{ is } B) \Rightarrow (x \text{ is } A)$.

The truth value of the proposition: "$Q\ B\ X$ are A" is then defined as:

$$\mu(Q\ B\ X\ are\ A) = \sup_{C \subseteq X} \min(\mu_{c'_1}(C),\ \mu_{c'_2}(C)) \quad (2)$$

with

$$\mu_{c'_1}(C) = \begin{cases} \mu_Q\left(\sum_{x \in C} \mu_B(x)\right) & \text{if } Q \text{ is absolute,} \\ \mu_Q\left(\dfrac{\sum\limits_{x \in C} \mu_B(x)}{\sum\limits_{x \in X} \mu_B(x)}\right) & \text{if } Q \text{ is relative} \end{cases} \quad (3)$$

and $\mu_{c'_2}(C) = \inf_{x \in C}\ \mu_B(x) \rightarrow \mu_A(x)$, where $\rightarrow$ is a fuzzy implication (see e.g. [7]).

c) Interpretation Based on the OWA Operator: In [19], Yager suggests to compute the truth degree of statements of the form "QBX are A" by an OWA aggregation of the implication values $\mu_B(x) \rightarrow_{KD} \mu_A(x)$ where $\rightarrow_{KD}$ denotes Kleene-Dienes implication ($a \rightarrow_{KD} b = \max(1 - a,\ b)$).

Let $X = \{x_1,\ \ldots,\ x_n\}$ such that $\mu_B(x_1) \leq \mu_B(x_2) \leq \ldots \leq \mu_B(x_n)$ and $\sum_{i=1}^{n} \mu_B(x_i) = d$. The weights of the OWA operator are defined by: $w_i = \mu_Q(S_i) - \mu_Q(S_{i-1})$, with $S_i = \sum_{j=1}^{i} \dfrac{\mu_B(x_j)}{d}$ and $S_0 = 0$. The implication values are denoted by c_i and ordered decreasingly: $c_1 \geq c_2 \geq \ldots \geq c_n$. Finally:

$$\mu(Q\ B\ X\ are\ A) = \sum_{i=1}^{n} w_i \times c_i. \quad (4)$$

III. Fuzzy Quantified statements in FURQL

In this section, we show how fuzzy quantified statements may be expressed in FURQL queries. We first propose a syntactic format for these queries, and then we show how they can be evaluated in an efficient way.

A. Syntax of a Fuzzy Quantified Query in FURQL

In the following, we consider fuzzy quantified statements of the type "$\mathcal{Q}\ B\ X$ are A" over fuzzy RDF graph databases, where the quantifier $\mathcal{Q}$ is represented by a fuzzy set and denotes either a relative quantifier (e.g., *most*) or an absolute one (e.g., *at least three*), B is the fuzzy condition "to be connected to a node x", X is the set of nodes in the graph, and A denotes a fuzzy (possibly compound) condition. An example of such a statement is: "*most* of the recent albums that are recommended by an artist, are highly rated and have been created by a young friend of this artist".

The general syntactic form of *fuzzy quantified queries* in the FURQL language is given in Listing 3.

```
DEFINE...
SELECT ?res WHERE {
  B(?res,?X)
  GROUP BY ?res
  HAVING Q(?X) ARE ( A(?X) ) }
```

Listing 3. Syntax of a FURQL quantified query R

The DEFINE clause allows to define the fuzzy terms and the fuzzy quantifier (denoted here by Q). The SELECT clause specifies which variables ?res should be returned in the result set, B(?res,?X) and A(?X) are SPARQL patterns such that the variables ?res and ?X appear in B, and ?X appears in A.

Example 4: The query, denoted by $R_{mostAlbums}$, that aims to retrieve every artist (?art1) such that *most* of the *recent* albums (?alb) that he/she recommends are *highly* rated and have been created by a *young* friend (?art2) of his/hers may be expressed in FURQL as follows:

```
1  DEFINEQRELATIVEASC most AS (0,1)
2  DEFINEASC high AS (2,5)
3  DEFINEDESC young AS (25,40)
4  DEFINEASC recent AS (2010,2015)
5  SELECT ?art1 WHERE {
6    ?art1 recommends ?alb. ?alb date ?date.
7    FILTER ( ?date IS recent ) }
8    GROUP BY ?art1
9    HAVING most(?alb) ARE
10       ( ?art1 friend ?art2. ?art2 creator ?alb.
11       ?alb rating ?rating. ?art2 age ?age.
12       FILTER (?rating IS high && ?age IS young) )
```

Listing 4. Syntax of the FURQL quantified query $R_{mostAlbums}$

where the DEFINEQRELATIVEASC clause defines the fuzzy relative increasing quantifier *most* of Figure 2.(c), the DEFINEASC clauses define the (increasing) membership functions associated with the fuzzy terms *high* and *recent* of Figure 2.(a) and (b), and the DEFINEDESC clause defines the (decreasing) membership function associated with the fuzzy term *young* of Figure 2.(d). In this query, ?art1 corresponds to ?res, ?alb corresponds to ?X, lines 6 to 7 correspond to B and lines 10 to 12 correspond to A. ◇

B. Evaluation of a Fuzzy Quantified Query

The interpretation of a fuzzy quantified statement in a FURQL query can be based on one of the formulas (1), (2), and (4). We first derive the original query into an intermediate query R_{flat} (given in Listing 5) aimed to retrieve the elements of the B part of the initial query, matching the variables ?res and ?X, for which we will then need to calculate the final satisfaction degree. For each pair (?res, ?X), we retrieve all the information needed for the calculation of μ_B and μ_A, i.e., the combination of fuzzy degrees associated with relationships and node attribute values involved in B(?res,?x) and in A(?X), respectively denoted by I_B and I_A.

```
SELECT ?res ?X I_B I_A WHERE {
 B(?res,?X)
 OPTIONAL { A(?X) } }
```

Listing 5. Derived query R_{flat} of $R_{mostAlbums}$

A simple post-processing step makes it possible to calculate the satisfaction degrees μ_B and μ_A according to I_B and I_A. If the optional part does not match a given answer, then $\mu_A = 0$. For the sake of simplicity, we consider in the following that the result of R_{flat} is made of the quadruples $(?\texttt{res}, ?\texttt{X}, \mu_B, \mu_A)$ matching the query.

Then, the answers to the initial fuzzy quantified query R (involving the fuzzy quantifier $\mathcal{Q}$) are answers to the query R_{flat} derived from R, and the final satisfaction degree associated with each answer e can be calculated according to the three different interpretations mentioned earlier in section II-C. Hereafter, we are going to illustrate this using Zadeh's approach [21] and Yager's OWA-based approach [19] (which are the most commonly used).

Following Zadeh's interpretation we have:

$$\mu(e) = \mu_{\mathcal{Q}}\left(\frac{\sum_{(m_i/?res=e)\in M_{R_{flat}}} \min(\mu_{A_i}, \mu_{B_i})}{\sum_{(m_i/?res=e)\in M_{R_{flat}}} \mu_{B_i}}\right) \quad (5)$$

In the case of a fuzzy absolute quantified query, the final satisfaction degree associated with each element e is simply $\mu(e) = \mu_{\mathcal{Q}}\left(\sum_{(m_i/?res=e)\in M_{R_{flat}}} \mu_{A_i}\right)$.

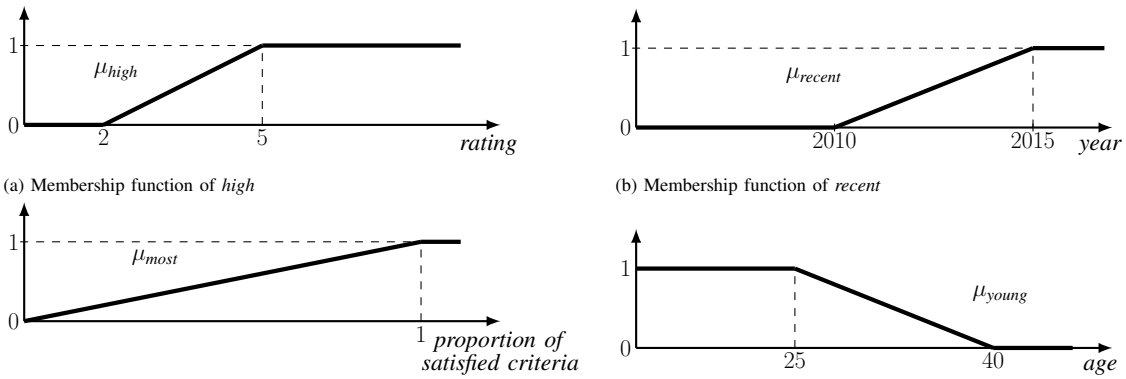
Example 5: Let us consider the query $R_{mostAlbums}$ of Listing 4. We evaluate this query according to the graph G_{MB} of Figure 1. In order to interpret $R_{mostAlbums}$, we first evaluate the query R_{flat} of Listing 6, derived from $R_{mostAlbums}$, that retrieves "the artists (?art1) who recommended at least one recent album (corresponds to B(?art1,?alb) in lines 2 and 3), possibly (OPTIONAL) highly rated and created by a young friend (corresponds to A(?alb) in lines 6 to 8)". This query returns a list of mappings of artist variables $(?art1)$ with their recommended albums $(?alb)$, satisfying the conditions of query R_{flat}, along with their respective satisfaction degrees $\mu_B = \min(\mu_{recent}(?alb), \rho_{recommends}(?art1, ?alb))$ and $\mu_A = \min(\mu_{high}(?rating), \mu_{young}(?age), \rho_{friend}(?art1, ?art2))$.

```
1  SELECT ?art1 ?alb μB μA WHERE {
2    ?art1 recommends ?alb. ?alb date ?date.
3    FILTER (?date IS recent)
4    OPTIONAL {
5    ?art1 friend ?art2. ?art2 creator ?alb.
6    ?alb rating ?rating. ?art2 age ?age.
7    FILTER (?rating IS high && ?age IS young) } }
```

Listing 6. Query R_{flat} derived from $R_{mostAlbums}$

In our example, R_{flat} concerns three artists X = {JustinT, Shakira, Beyonce}. EnriqueI, Drake, Mariah and Rihanna do not belong to the result set of R_{flat} because EnriqueI, Drake and Mariah have not recommended any album made by any of their friends and Rihanna did not recommend any somewhat recent album. Then $M_{R_{flat}}$ = {(?art1→ JustinT, ?alb→ One dance, $\mu_B \to 0.4$, $\mu_A \to 0.3$), (?art1→ JustinT, ?alb → Home, $\mu_B \to 0.1$, $\mu_A \to 0.6$), (?art1→ Shakira, ?alb → Euphoria, $\mu_B \to 0.1$, $\mu_A \to 0.07$), (?art1→ Shakira, ?alb → Butterfly, $\mu_B \to 0.2$, $\mu_A \to 0$), (?art1→ Shakira, ?alb → Justified, $\mu_B \to 0.3$, $\mu_A \to 0.4$), (?art1→ Beyonce, ?alb → Home, $\mu_B \to 0.4$, $\mu_A \to 0.3$)}. Finally, assuming for the sake of simplicity that $\mu_{most}(x) = x$, the final result of the query $R_{mostAlbums}$ evaluated on G_{MB} using Formula 5 is: {0.80/JustinT, 0.75/Beyonce, 0.62/Shakira}. ◇

Using Yager's OWA-based approach, for each element e returned by R_{flat} we calculate $\mu(e) = \sum_{(x_i/?res=e)\in M_{R_{flat}}} w_i \times c_i$. Let us consider condition $B = \{\mu_{B_1}/x_1, ..., \mu_{B_n}/x_n\}$ such that $\mu_{B_1} \leq ... \leq \mu_{B_n}$, condition $A = \{\mu_{A_1}/x_1, ..., \mu_{A_n}/x_n\}$ and $d = \sum_{i=1}^{n} \mu_{B_i}$.



(a) Membership function of *high*

(b) Membership function of *recent*

(c) Membership function of *most*

(d) Membership function of *strong*

Figure 2. Membership functions

The weights of the *OWA* operator are defined by $w_i = \mu_Q(S_{x_i}) - \mu_Q(S_{x_{i-1}})$ with $S_{x_i} = \sum_{j=1}^{i} \frac{\mu_{B_j}}{d}$ and the implication values are denoted by $c_{x_i} = max(1 - \mu_{B_i}, \mu_{A_i})$ and ordered decreasingly such that $c_1 \geq \ldots \geq c_n$.

Example 6: In order to calculate $\mu(\texttt{Shakira})$ from R_{flat}, let us consider B (resp. A) the set of satisfaction degrees corresponding to condition B (resp. A) of element `Shakira` as follows B ={ 0.1/`Euphoria`, 0.2/`Butterfly`, 0.3/`Justified`} and A= { 0.07/`Euphoria`, 0/`Butterfly`, 0.4/`Justified`}. We have $d = 0.6$ and $S_{Euphoria} = \frac{0.1}{0.6} = 0.17$, $S_{Butterfly} = \frac{0.1+0.2}{0.6} = 0.5$, and $S_{Justified} = \frac{0.1+0.2+0.3}{0.6} = 1$.

Then, with $\mu_{most}(x) = x$, we get $\mu_Q(S_{Euphoria})$ = 0.17, $\mu_Q(S_{Butterfly})$ = 0.5 and $\mu_Q(S_{Justified})$ = 1. Therefore, the weights of the OWA operator are $W_1 = \mu_Q(S_{Euphoria}) - \mu_Q(S_0) = 0.17$, $W_2 = \mu_Q(S_{Butterfly}) - \mu_Q(S_{Euphoria}) = 0.33$, and $W_3 = \mu_Q(S_{Justified}) - \mu_Q(S_{Butterfly}) = 0.5$.

The implication values are $c_{Euphoria} = \max(1-0.1, 0.07) = 0.9$, $c_{Butterfly} = \max(1-0.2, 0) = 0.8$, and $c_{Justified} = \max(1-0.3, 0.36) = 0.7$.

Thus, $c_1 = 0.9$, $c_2 = 0.8$ and $c_3 = 0.7$. Finally, we get $\mu(\texttt{Shakira}) = 0.17 \times 0.9 + 0.33 \times 0.8 + 0.5 \times 0.7 = 0.15 + 0.26 + 0.35 = 0.77$. $\diamond$

IV. IMPLEMENTATION

The evaluation strategy we propose for processing these queries consists of a software add-on layer over a standard classical SPARQL engine. This software, called SURF[3] (Sparql with fUzzy quantifieRs for rdF data), is implemented within the Jena Semantic Web Java Framework[4] for creating and manipulating RDF graphs. SURF evaluates FURQL queries that contain fuzzy quantified statements whose syntax was presented above. It basically consists of two modules.

1) A pre-processing module, the *query compiler*, produces (i) the query dependent functions that allow us to compute μ_B, μ_A and μ, for each returned answer, according to the chosen interpretation, and (ii) a (crisp) SPARQL query R_{flat}, which is then sent to the SPARQL query engine for retrieving the information needed to calculate μ_B and μ_A. The compilation uses the derivation principle introduced in [2].
2) A post-processing module first performs a `GROUP BY` of the elements and, then, calculates μ_B, μ_A and μ for each returned answer, ranks the answers, and filters them if an α-cut has been specified in the initial fuzzy query.

Figure 3 illustrates this architecture.

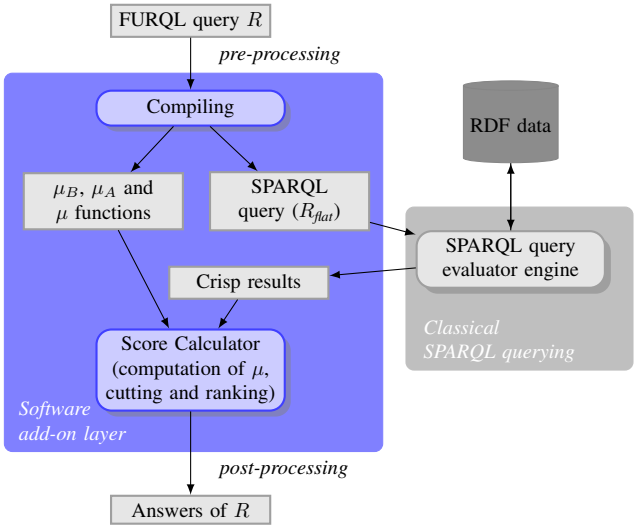


Figure 3. SURF software architecture

For quantified queries of the type "Q B X are A", the principle is to first evaluate the fuzzy query R_{flat} derived from the original query. For each tuple x from the result of R_{flat}, we return the satisfaction degrees related to conditions A and B, denoted respectively by μ_A and μ_B. The final satisfaction degree μ can be calculated according to Formulas (1), (2) or (4) using the value of μ_B and μ_A. At the current time, Zadeh's approach [21] and Yager's OWA-based approach [19] have been implemented, and the choice of the interpretation to be used is made through the configuration tool of the system. Finally, we get a set of answers ranked in decreasing order of their satisfaction degree.

[3]https://www-shaman.irisa.fr/surf/

[4]https://jena.apache.org

Table 1
SET OF FUZZY QUANTIFIED QUERIES

Query	P_B	P_A	Conditions
$Q1_{crisp}$	simple	simple	crisp
$Q2_{crisp}$	complex	simple	crisp
$Q3_{crisp}$	simple	complex	crisp
$Q4_{crisp}$	complex	complex	crisp
$Q1_{fuzzy}$	simple	simple	fuzzy
$Q2_{fuzzy}$	complex	simple	fuzzy
$Q3_{fuzzy}$	simple	complex	fuzzy
$Q4_{fuzzy}$	complex	complex	fuzzy

V. EXPERIMENTAL RESULTS

In order to demonstrate the performances of our approach in the case of selection SPARQL graph pattern queries, we run some experiments. We considered a set of fuzzy quantified queries divided in two types: Q_{crisp} and Q_{fuzzy} where Q_{crisp} (resp. Q_{fuzzy}) is a fuzzy quantified query involving crisp conditions (resp. fuzzy conditions). We processed four crisp and four fuzzy queries by changing each time the nature of the patterns corresponding to condition B and A from simple to complex ones. A complex pattern differs from a simple one by the number and the nature (including structural properties) of its statements. These queries are summarized in Table 1.

Our RDF data is inspired by Musicbrainz linked data (which is originally crisp), and for representing fuzzy information, we used the reification mechanism that makes it possible to attach fuzzy degrees to triples, as proposed in [22].

For these experiments, we used four different sizes of fuzzy RDF datasets containing crisp and fuzzy triples (DB_1 of 11796 triples, DB_2 of 65994 triples, DB_3 of 112558 triples and DB_4 of 175416 triples. The results according to Yager's OWA-based interpretation are depicted in Figure 4. Figure 4 presents the execution time in milliseconds of the processing of the fuzzy quantified queries involving fuzzy conditions from Table 1 over the considered RDF datasets (due to space limitations, results for queries involving crisp conditions are note given but are quite similar). All experiments were carried out on a computer running Windows 7 (64 bits) with 8GB of RAM.

An important result is that, for all the fuzzy quantified queries involving fuzzy and crisp conditions presented in Table 1, the processing time of the overall process is proportional to the size of the dataset. It is especially striking to see that the processing time taken by the compiling step and the score calculation step, which are directly related to the introduction of flexibility into the query language, are very strongly dominated by the time taken by the SPARQL evaluator (which includes the time for executing the query and getting the result set).

Moreover, The FURQL compiling step takes so little time compared to the other two steps that it cannot even be seen in Figure 4. This time remains almost constant, and is independent on the size of the dataset while slightly increasing with complex patterns or fuzzy conditions. As to the score calculation step, it represents around 10% of the time needed for evaluating a crisp SPARQL query. The time used for calculating the final satisfaction degree is of course dependent on the size of the result and the nature of the patterns.

Finally, these results show that introducing fuzzy quantified statements into a SPARQL query entails a very small increase of the overall processing time.

This conclusion can obviously be extended to the case of Zadeh's interpretation, inasmuch as it is even more straightforward than Yager's OWA-based approach. Thus, the processing time of the score calculating step can only be smaller than in the case of Yager's OWA-based interpretation.

VI. RELATED WORK

In a graph database context, there have been some recent proposals for incorporating quantified statements into user queries. In [3], Bry et al. propose an extension of SPARQL (called SPARQLog) with first-order logic (FO) rules and existential and universal quantification over node variables. This query language makes it possible to express statements such as: *"for each lecture there is a course that practices this lecture and is attended by all students attending the lecture"*.

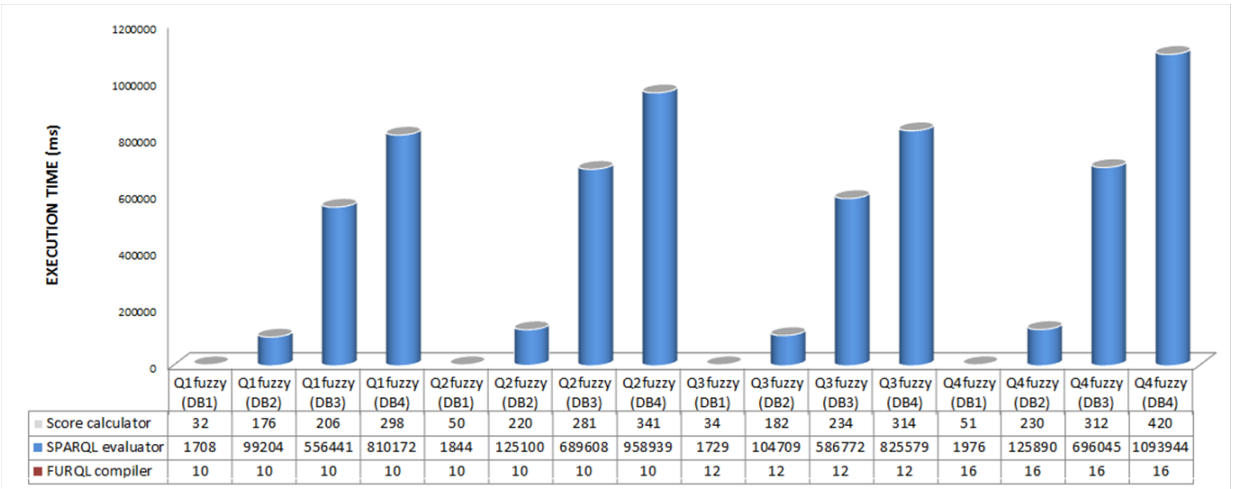
More recently, in [6], Fan et al. introduced quantified graph patterns (QGPs), an extension of the classical SPARQL graph patterns using simple counting quantifiers on edges. On the other hand, quantified graph patterns make it possible to express numeric and ratio aggregates, and negation besides existential and universal quantification. The authors also showed that quantified matching in the absence of negation does not significantly increase the cost of query processing.

However, to the best of our knowledge, there does not exist any work in the literature that deals with fuzzy quantified patterns in the SPARQL query language, which was the main goal of our work. Fuzzy quantified queries have been long studied in a relational database context, see e.g. [1] whose authors distinguish two types of fuzzy quantification: horizontal quantification [9] used for combining atomic conditions in a *where* clause and vertical quantification for which the quantifier appears in a *having* clause in order to express a condition on the cardinality of a fuzzy subset of a group. This is the type of use we make in our approach.

In a graph database context, fuzzy quantified queries have an even higher potential since they can exploit the structure of the graph, beside the attribute values attached to the nodes or edges (see, [20], [4], [5], and [14]). The work the most related to that presented here is [14], where we considered a particular type of fuzzy quantified structural statement in the general context of fuzzy graph databases. An example of such a statement is: *"most of the recent papers of which x is a main author, have been published in a renowned database journal"*. We showed how this statement could be expressed in the language FUDGE (which is a flexible extension of the CYPHER query language) that we previously proposed [15].

VII. CONCLUSION

In this paper, we have investigated the issue of integrating fuzzy quantified queries of the type "$Q\ B\ X$ are A" into a language aimed to query fuzzy RDF graph databases. We



	Q1fuzzy (DB1)	Q1fuzzy (DB2)	Q1fuzzy (DB3)	Q1fuzzy (DB4)	Q2fuzzy (DB1)	Q2fuzzy (DB2)	Q2fuzzy (DB3)	Q2fuzzy (DB4)	Q3fuzzy (DB1)	Q3fuzzy (DB2)	Q3fuzzy (DB3)	Q3fuzzy (DB4)	Q4fuzzy (DB1)	Q4fuzzy (DB2)	Q4fuzzy (DB3)	Q4fuzzy (DB4)
Score calculator	32	176	206	298	50	220	281	341	34	182	234	314	51	230	312	420
SPARQL evaluator	1708	99204	556441	810172	1844	125100	689608	958939	1729	104709	586772	825579	1976	125890	696045	1093944
FURQL compiler	10	10	10	10	10	10	10	10	12	12	12	12	16	16	16	16

Figure 4. Fuzzy Quantified queries involving fuzzy conditions over different size of DB

have defined the syntax and semantics of an extension of the FURQL query language, that makes it possible to deal with such queries. A query processing strategy based on the derivation of non-quantified fuzzy queries has also been proposed, and we performed some experiments in order to study its performances. The results of these experiments show that the extra cost induced by the fuzzy nature of the queries remains very limited, even in the case of rather complex fuzzy quantified queries such as those considered in this approach.

As a future work, we plan to study other types of (more complex) fuzzy quantified queries, in particular, those that aim to find the nodes x such that x is connected (by a path) to Q nodes reachable by a given pattern and satisfying a given condition C.

Acknowledgement: This work has been partially funded by the French DGE (Direction Générale des Entreprises) under the project ODIN (Open Data INtelligence).

REFERENCES

[1] P. Bosc, L. Liétard, and O. Pivert. Quantified statements and database fuzzy querying. In P. Bosc and J. Kacprzyk, editors, *Fuzziness in Database Management Systems*, pages 275–308. Physica Verlag, 1995.

[2] P. Bosc and O. Pivert. Sqlf query functionality on top of a regular relational database management system. In *Knowledge Management in Fuzzy Databases*, pages 171–190. Springer, 2000.

[3] F. Bry, T. Furche, B. Marnette, C. Ley, B. Linse, and O. Poppe. SPARQLog: SPARQL with rules and quantification. In *Semantic Web Information Management*, pages 341–370. Springer, 2010.

[4] A. Castelltort and A. Laurent. Fuzzy queries over NoSQL graph databases: Perspectives for extending the cypher language. In *Proc. of IPMU'14*, pages 384–395, 2014.

[5] A. Castelltort and A. Laurent. Extracting fuzzy summaries from NoSQL graph databases. In *Proc. of the Intl. Conf. on Flexible Query Answering Systems (FQAS'15)*, pages 189–200, 2015.

[6] W. Fan, Y. Wu, and J. Xu. Adding counting quantifiers to graph patterns. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1215–1230. ACM, 2016.

[7] J. Fodor and R. Yager. Fuzzy-set theoretic operators and quantifiers. In D. Dubois and H. Prade, editors, *The Handbooks of Fuzzy Sets Series, vol. 1: Fundamentals of Fuzzy Sets*, pages 125–193. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[8] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, 2013. http://www.w3.org/TR/sparql11-query.

[9] J. Kacprzyk, S. Zadrożny, and A. Ziółkowski. FQUERY III +:a "human-consistent" database querying system based on fuzzy logic with linguistic quantifiers. *Inf. Syst.*, 14(6):443–453, 1989.

[10] Y. Lv, Z. Ma, and L. Yan. Fuzzy RDF: A data model to represent fuzzy metadata. In *FFUZZ-IEEE 2008*, pages 1439–1445, June 2008.

[11] M. Mazzieri and A. Dragoni. A fuzzy semantics for the resource description framework. In P. da Costa, C. dAmato, N. Fanizzi, K. Laskey, K. Laskey, T. Lukasiewicz, M. Nickles, and M. Pool, editors, *Uncertainty Reasoning for the Semantic Web I*, volume 5327 of *Lecture Notes in Computer Science*, pages 244–261. Springer Berlin Heidelberg, 2008.

[12] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.

[13] O. Pivert, O. Slama, and V. Thion. An extension of SPARQL with fuzzy navigational capabilities for querying fuzzy RDF data. In *IEEE International Conference on Fuzzy Systems*, 2016.

[14] O. Pivert, O. Slama, and V. Thion. Fuzzy quantified structural queries to fuzzy graph databases. In *International Conference on Scalable Uncertainty Management*, pages 260–273. Springer, 2016.

[15] O. Pivert, V. Thion, H. Jaudoin, and G. Smits. On a fuzzy algebra for querying graph databases. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 748–755. IEEE, 2014.

[16] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recomm., 2008.

[17] W3C. RDF overview and documentations, 2014. http://www.w3.org/RDF/.

[18] R. R. Yager. General multiple-objective decision functions and linguistically quantified statements. *International Journal of Man-Machine Studies*, 21:389–400, 1984.

[19] R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, 1988.

[20] R. R. Yager. Social network database querying based on computing with words. In O. Pivert and S. Zadrożny, editors, *Flexible Approaches in Data, Information and Knowledge Management*, volume 497 of *Studies in Computational Intelligence*, pages 241–257. Springer, 2013.

[21] L. A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Comput. and Math. with Applications*, 9:149–183, 1983.

[22] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semant.*, 11:72–95, Mar. 2012.